FPGA-Based Trigger for LArIAT

(last updated 9/29/2016)

A firmware trigger system for the LArIAT experiment, targeting the CAEN V1495 User FPGA (Altera Cyclone I EP1C20F400C6).

The firmware may be useful as a starting point for other similar projects. The VHDL code originated with an example provided by CAEN S.p.A. but has been heavily modified and extended. The firmware is not a CAEN product and is not supported by CAEN.

Abstract

Digitizer modules require a "trigger" signal to know when to record data. In the case of LArIAT, this data is from the TPC. An XML configuration file (parsed and written into FPGA registers) specifies the conditions (a coincidence of signals from beamline detectors) under which good events are likely to occur. When the input channel signals match this, a trigger is released.

Features

- 32 input buses
- Fast trigger (~80 ns delay from signal rising edge)
- Slow trigger (programmable delay)
- Veto mechanism to avoid pileups (programmable veto time)
- 16 trigger coincidence conditions, 16 veto coincidence conditions
- Prescaling for the 16 trigger conditions
- Pattern and timestamp FIFO pipelines, capacity 4096 entries
- Majority logic (M of N) for the four wire chamber channels
- Global clock running at 100 MHz, giving 10 ns signal resolution

Modules

Firmware Authors

- Matthew Stephens (William and Mary)
- James Zhu (UC Berkeley) - james.zhu.engineer@gmail.com

The firmware is, substantially, the senior thesis work of Matthew Stephens, carried out in 2013-14. See mjstephens_thesis.pdf (docdb 1287) for technical details. This project was conducted with support from the National Science Foundation under award #0855526.

Advisors

- Mike Kordosky, Ph.D, Associate Professor of Physics (William and Mary) - makordosky@wm.edu
- Will Flanagan, Ph.D (U. of Texas at Austin) - will.flanagan@utexas.edu
- William Badgett, Ph.D - badgett@fnal.gov
- Jason St. John, Ph.D (U. Cincinnati) - stjohn@fnal.gov

License

Developing

Requirements

System Requirements

The FPGA on the V1495 is an Altera Cyclone (1st gen). Altera's toolchain has the following requirements:

- OS: Windows or Linux (OS X unsupported)
- Disk Space: ~ 4 GB
- Linux system libraries (32-bit versions):
  - glibc
  - libXext
  - libX11
  - libXau
  - libXdmcp
  - freetype
  - fontconfig
  - expat

Note: ModelSim-Altera is only compatible with ncurses v5, because of a breaking ABI in v6.

Compiler

The last versions of Quartus, Altera's FPGA design tool, to support this are:

- **Quartus II Web Edition** 11.0 SP1 (free version)
  - https://wl.altera.com/download/software/quartus-ii-we/11.0sp1
- **Quartus II Subscription Edition** 13.0 SP1 (paid version)
  - https://dl.altera.com/13.0sp1/

Simulation

Altera's FPGA simulation tool, ModelSim-Altera, corresponds closely to the version of Quartus you are using:

- **ModelSim-Altera Starter** v6.6d SP1 for Quartus II v11.0 SP1
  - https://wl.altera.com/download/software/modelsim-starter/11.0
- **ModelSim-Altera** (included with Quartus 13.0 download)
  - https://dl.altera.com/13.0sp1/

The following install instructions assume the use of Quartus II Web Edition (11.0SP1) and ModelSim-Altera Starter (6.6d), with some form of Linux.

Installation on Windows is similar, if not almost identical.

Installation

The downloads will appear as the following:

```
11.0sp1_quartus_free_linux.sh
11.0sp1_modelsim_ase_linux.sh
```

Give yourself permissions to run the scripts, i.e.

```
chmod +x 11.0sp1_quartus_free_linux.sh
chmod +x 11.0sp1_modelsim_ase_linux.sh
```

and then execute them:

```
./11.0sp1_quartus_free_linux.sh
./11.0sp1_modelsim_ase_linux.sh
```

For each one, a GUI will appear on your screen, directing you to select the install directory and which device drivers to install.

Coding for the V1495 only requires that the "Cyclone" drivers be installed ("Cyclone II", "Cyclone III", "Cyclone IV", and "Cyclone V" are not required).

The Quartus Web Edition shouldn't require a license, so the popup asking for a license can be ignored.

This should take about 15-30 minutes to install.

Compiling

The Quartus project file is `fpga/lariat-trigger.qpf`. Open it from the File menu and you should have the files and device config loaded for you.

Compiling is done by double-clicking the "Compile Design" arrow under the "Tasks" pane. There are several intermediary parts: - **Analysis and Synthesis** (1 min): syntax & logic checking. Useful for performing a quick REPL cycle. - **Fitter (place & route)** (3-4 min): maps the logic gates to the physical device. - **Assembler** (30 s): generates the final binary. - **TimeQuest Timing Analysis** (30 s): checks if device performs within timing requirements. - **EDA Netlist Writer** (15 s): generates connectivity files for other tooling. *Currently unused.*

The full compile cycle will generate the files `fpga/output_files/output_file.sof`. To turn this into the final raw binary file (.rbf), perform the following steps: - Open "File" -> "Convert Programming Files". - Click "Open Conversion Setup Data" -> select `fpga/rbf_setup.cof`. - Provide a name for the raw binary file. - Click "Generate".

This can then be copied to the lariat-daq00 server via `scp`.

Upgrading V1495 Firmware

See also section 5.9 of the V1495 spec for more details.

- SSH into `lariat-daq00.fnal.gov` as the `lariat` user.
- Run `CAENUpgraderGUI`.
- Fill out the forms as below:
  - Available Actions: Upgrade Firmware
  - Board Model: V1495
  - Firmware binary file:
  - Connection Type: OPTLINK
  - LINK number: 0
  - Board number: 0
  - VME Base Address: `0x04800000`
- **Make sure User FPGA, not VME FPGA is selected.**
  - Selecting the wrong FPGA will render the board inaccessible through VME.
  - If this happens, the VME FPGA should have a backup firmware. Consult section 5.9.1. of the V1495 spec (on docdb) to fix it.
  - If the backup firmware fails, consult section 5.9.2 of the V1495 spec to fix it. (However, the recommended solution was to send the board to CAEN for a repair.)
- Click "Upgrade".
- Close the upgrade tool and run `lariatReset` to power cycle the board.

Verifying V1495 Firmware

Instead of selecting **Upgrade Firmware** in the CAEN Upgrader GUI, select **Verify Firmware**. This will verify that the loaded firmware matches the firmware selected in the dialog box.

Simulation

Simulation is useful for verifying the firmware's functionality, but it cannot help in debugging bad register reads / writes over the VME (i.e. through V1495Driver.cc).

A test bench is provided at `fpga/src/trigger_top_mini.vht`. This must be compiled separately, but offers a scaled-down version of the trigger_top module to reduce complexity.

Simulation requires only that the **Analysis and Synthesis** phase of compilation be complete. After that, navigate to

> Tools -> Run EDA Simulation Tool -> EDA RTL Simulation

to compile, and simulate the test bench.

Simulating individual modules can be accomplished by expanding the **work** library and Right-Click -> Simulate any module.

Compile errors will appear in the bottom message pane.

Troubleshooting

> The code takes forever to compile.

A full compile for trigger_top_K took about 5 minutes on my Thinkpad T430.

If *"Fitter routing failed. Retrying ..."* appears in the processing log, the design is too complex to fit on the device with a simple fit, forcing the compiler to optimize the design more aggressively in a second pass. Fixing this requires simplifying the logic.

Bitcounter

```
ENTITY bitcounter is
  port(
    LCLK     : in  std_logic;
    reset    : in  std_logic;
    bitIn    : in  std_logic;
    outWidth : in  std_logic_vector(1 downto 0);

    bitcount : out std_logic_vector(31 downto 0);
    outPulse : out std_logic
);
END bitcounter;
```

`fpga/src/bitcounter.vhd`

This module performs two tasks: - Synchronizing pulses from a single `raw_input` channel into `sync_input` - Counting signals on a channel specified by `bitIn`

These synchronized pulses are of uniform length (specified in clock ticks by `outWidth`), synchronized to the global clock `LCLK`, and emitted in `outPulse`.

Inputs

Name	Description
`LCLK`	the global clock, running at 100 MHz (10 ns period).
`reset`	the global reset (active low).
`bitIn`	the channel to synchronize, as sliced from `raw_input`.
`outWidth`	the width of the synchronized signal.

Outputs

Name	Description
`bitcount`	the number of signals counted on this channel. Overflows at 2^32 - 1.
`outPulse`	the synchronized signal, as sliced into `sync_input`.

Limitations

Signals shorter than the global clock period (10 ns) may be lost if they do not occur during the global clock's rising edge.

This results in the trigger system's resolution of 10ns.

M of N

```
ENTITY MofN is
  port(
    a   : in std_logic;
    b   : in std_logic;
    c   : in std_logic;
    d   : in std_logic;
    M   : in std_logic_vector(2 downto 0);
    O   : out std_logic
);
END MofN;
```

`fpga/src/mofn.vhd`

This module implements M of N logic for four channels (N = 4), i.e. the output is set high when M or more of the four channels are high. If M is set to an invalid value (i.e. not between 1 and 4), then the output is set always set to low.

Inputs

Name	Description
`a`	signal a (active high)
`b`	signal b (active high)
`c`	signal c (active high)
`d`	signal d (active high)
`M`	number of signals needed for output signal

Outputs

Name	Description
`O`	output signal (active high)

Limitations

Coincidence

```
ENTITY coincidence is
  port(
    LCLK        : in  std_logic;
    reset       : in  std_logic;
    bitsIn      : in  std_logic_vector(31 downto 0);
    bitMask     : in  std_logic_vector(31 downto 0);
    testPattern : in  std_logic_vector(31 downto 0);
    prescaleCnt : in  std_logic_vector(31 downto 0);

    patternCnt  : out  std_logic_vector(31 downto 0);
    patternIn   : out  std_logic
  );
END coincidence;
```

`fpga/src/coincidence.vhd`

This module performs the essential logic of the trigger firmware - checking for a specific pattern of signals from specific channels (detectors).

Specifically, this module masks the incoming signals `bitsIn` against `bitMask` and checks for a match against `testPattern`. If there is a match, then a signal is sent to `patternIn`.

As of `trigger_top_K`, this module now has prescaler capabilities, allowing the module to output a signal only after "n" matching patterns.

The total number of pattern matches (after prescaling) is written to `patternCnt`.

Inputs

Name	Description
`LCLK`	the global clock, running at 100 MHz (10 ns period).
`reset`	the global reset (active low).
`bitsIn`	the channel signals, passed in from `channels` in trigger_top.
`bitMask`	the signals to check and ignore, stored in a register.
`testPattern`	the signals to match, stored in a register.
`prescaleCnt`	the prescaler ratio, stored in a register.

Outputs

Name	Description
`patternCnt`	the total number of pattern matches.
`patternIn`	the signal indicating a match (active high, one tick pulses).

Limitations

- `patternCnt` overflows after (2^32 - 1) matches.
- A signal is only emitted on the rising edge, i.e. if the correct pattern of channels is continuously matched, then no new signal is emitted.

Delay

```
ENTITY delay is
  port(
    LCLK          : in  std_logic;
    reset         : in  std_logic;
    delayLength   : in  std_logic_vector(31 downto 0);
    triggerWidth  : in  std_logic_vector(1 downto 0);
    goodPattIn    : in  std_logic;

    busy          : out std_logic;
    triggerOut    : out std_logic
);
END delay;
```

`fpga/src/delay.vhd`

This module implements a programmable delay between an incoming recognized pattern (i.e. the fast trigger) and the slow trigger signal.

During the delay, any additional fast triggers are ignored. If a reset signal (from either the global reset or the veto module) interrupts a delay, no slow trigger is emitted.

Inputs

Name	Description
`LCLK`	the global clock, running at 100 MHz (10 ns period).
`reset`	whether to interrupt the slow trigger (global reset or veto) (**active high**).
`delayLength`	the delay before emitting the slow trigger (in clock ticks).
`triggerWidth`	the width of the slow trigger signal (in clock ticks).
`goodPattIn`	the fast trigger (which starts the countdown to the slow trigger).

Outputs

Name	Description
`busy`	indicates a delay or a slow trigger is occurring
`triggerOut`	the slow trigger signal

Limitations

Veto behavior requires further study.

Veto

```
ENTITY veto is
  port(
    LCLK          : in  std_logic;
    reset         : in std_logic;
    vetoLength    : in  std_logic_vector(31 downto 0);
    vetoIn        : in  std_logic;

    busy          : out std_logic
);
END veto;
```

`fpga/src/veto.vhd`

This module generates veto signals, which is fed into the delay module to interrupt slow triggers.

Veto signals are generated on some of certain conditions: - A good pattern is matched during a slow trigger's delay - A good pattern is matched during another veto signal - A bad pattern is matched

The veto signal may also be referenced in other documentation / code as a veto delay (notably in the V1495 config).

Inputs

Name	Description
`LCLK`	the global clock, running at 100 MHz (10 ns period).
`reset`	the global reset (active low).
`vetoLength`	the width of the veto signal (in clock ticks).
`vetoIn`	whether to generate a veto signal or not.

Outputs

Name	Description
`busy`	the veto signal.

Limitations

Veto behavior requires further study. See docdb-1612 for more info.

To Digitizer

```
ENTITY to_digitizer is
  port(
    LCLK          : in  std_logic;
    reset         : in  std_logic;
    patternVector : in  std_logic_vector(15 downto 0);
    delayLength   : in  std_logic_vector(31 downto 0);

    output        : out std_logic_vector(15 downto 0)
);
END to_digitizer;
```

`fpga/src/todigitizer.vhd`

When a good pattern is detected, this module gathers two pieces of information to send to the V1740 digitizers along with the slow trigger: - The good pattern detected - How many good patterns have been detected (total)

Inputs

Name	Description
`LCLK`	the global clock, running at 100 MHz (10 ns period).
`reset`	the global reset (active low).
`patternVector`	the array of patterns detected (from the coincidence modules).
`delayLength`	the slow trigger delay time. Used to sync the digitizer signal with the slow trigger.

Outputs

Name	Description
`output`	the below outputs concatenated into a single vector.
`output(11 downto 0)`	The total number of slow triggers outputted.
`output(15 downto 12)`	The pattern's ID (0 - 15).

Limitations

This module only expects one pattern to be detected at a time. If multiple patterns are detected, the first (lowest index) pattern is transmitted.

Timestamp

```
ENTITY timestamp is
  port(
    LCLK          : in  std_logic;
    reset         : in  std_logic;
    timestamp_out : out std_logic_vector(31 downto 0)
);
END timestamp;
```

`fpga/src/timestamp.vhd`

This module maintains a timestamp on `timestamp_out` that increments every clock cycle. This is used in the timestamp pipeline (FIFO) to uniquely identify patterns.

Inputs

- `LCLK` - the global clock, running at 100 MHz (10 ns period).
- reset` - the global reset (active low).

Outputs

- `timestamp_out` - the current timestamp.

Limitations

The timestamp overflows at (2^32 - 1) clock ticks = 42.9 seconds. This is just long enough for every pattern in a 30-second spill to be uniquely identified before the system is reset.

FIFOs (Pattern & Timestamp)

```
ENTITY fifo_4Kx32 is
  port(
    LCLK          : in std_logic;
    reset         : in std_logic;
    data_in       : in std_logic_vector(31 downto 0);
    store_data    : in std_logic;
    read_data     : in std_logic;
    set_read      : in std_logic;

    num_entries   : out std_logic_vector(31 downto 0);
    data_out      : out std_logic_vector(31 downto 0)
);
END fifo_4Kx32;
```

`fpga/src/fifo_4Kx32.vhd`

This module creates a FIFO queue, backed by emulated RAM from the FPGA's embedded memory. It is used to maintain a queue of patterns and their corresponding timestamps, ordered by oldest first.

The FIFO is a finite state machine, with the following states: FULL: - IDLE: await `store_data` or `read_data` signals. update read and write RAM addresses to register values - WRITING: move address to write position, store current value of data_in - READING: move address to read position, send data to data_out. Increment read position. - FULL: RAM is completely full. `store_data` is ignored. `read_data` transitions to READING.

Patterns / timestamps are stored on each good pattern.

Patterns / timestamps are read when the pattern / timestamp registers are read When the pattern and timestamp registers are read from the VME (REG_R50 and REG_R51), `read_data` is set to true. from the VME (REG_R50 and REG_R51).

When the number of FIFO entries is read from the VME (REG_R49), `set_read` is sent a signal. (I'm not certain of the specifics beyond there.)

Inputs

Name	Description
`LCLK`	the global clock, running at 100 MHz (10 ns period).
`reset`	the global reset (active low).
`data_in`	the data to store (push) into the queue.
`store_data`	flag indicating whether to store (push) an entry or not.
`read_data`	flag indicating whether to read (pop) an entry or not.
`set_read`	switch between manipulating the read or write addresses

Outputs

Name	Description
`num_entries`	number of stored entries in the FIFO.
`data_out`	the data read out if `read_data` is set.

Limitations

The FIFO can only store 4096 entries (i.e. 4096 patterns). Any patterns stored after it is FULL will be lost.

VME Interface

```
ENTITY LB_INT is
  port(
    -- Local Bus in/out signals
    nLBRES     : in     std_logic;
    nBLAST     : in     std_logic;
    WnR        : in     std_logic;
    nADS       : in     std_logic;
    LCLK       : in     std_logic;
    nREADY     : out    std_logic;
    nINT       : out    std_logic;
    set_read   : out    std_logic;
    read_fifo  : out    std_logic_vector(1 downto 0);
    LAD        : inout  std_logic_vector(15 downto 0);

    -- Internal Registers
    REG_R   : in reg_vector(67 downto 0);
    REG_RW : buffer reg_vector(83 downto 0)
  );

END LB_INT;
```

`fpga/src/lb_int.vhd`

This module allows registers on the V1495 User FPGA to be read and written from the V2718 optical link bridge through the VME interface.

Two kinds of registers are used: read-only and read-write registers, indicating the level of access from the VME interface.

(These registers are all readable and writable internally.)

There are four sections of note: 1. Address list: the addresses the registers are mapped to 2. Initialization: the values of the registers when they are being reset 3. Reading: reading the read-only and read-write registers 4. Writing: writing to the read-write registers.

When adding / removing registers, all 4 of these must be updated. This process is automated with the `update_registers.py` script, which reads `registertable.txt` and `lb_int.template` to generate a new `lb_int.vhd` file.

Inputs

Name	Description
`LCLK`	the board's global clock (not the external clock), at 40 MHz.
`nLBRES`	(Local Bus RESet)
`nBLAST`	(something LAST)

Name	Description
`WnR`	(Write not Read) bit indicating write if true, read if false
`nADS`	(something) used to start a read or write cycle (pulled low)
`REG_R`	Read-only registers, stored as signals in `trigger_top`.
`REG_RW`	Read-write registers, stored as signals in `trigger_top`.

Outputs

Name	Description
`nREADY`	something to do with ready to read from the VME?
`nINT`	Unused?
`set_read`	Signal to read the FIFO.
`read_fifo`	Signal indicating whether pattern or timestamp FIFO is to be read.
`LAD`	LAD bus, used to transfer address from VME to FPGA and register contents from FPGA to VME

Limitations

CAEN V1495 Spec v16, § 5.1

- Register addresses are limited to the range `0x1000` to `0x7FFF`.
- `0x000C` (mapped at `0x100C` over the VME bus) MUST be implemented and readable. The “Bridge” FPGA reads this address to decide whether to enable the time-bomb mechanism on the running firmware or not. Particularly, bit[15] of the register MUST be 0 in order the time-bomb is not activated.

CAEN V1495 Spec v16, § 5.4.2 - Registers are 32 bits large. (`lb_int` is designed for only 32-bit read / writes.) - Registers may be read-only, write-only, or read/write. (We use only read-only and read-write registers, as write-only registers are hard to debug.) - Default values: - Read/write: value after a reset - Write-only: value after a reset - Read-only: status of signals read by the FPGA and have no default value. (Not sure what "status of signals" means in the spec.)

Signal specifics

- Signals:
  - REG_WREN and REG_RDEN: one-clock-cycle pulses enabling a write / read access to a register.
  - REG_ADDR: register address
  - Writing into a register:
  - Data is available, 16 bits at at a time, through REG_DIN.
  - Guaranteed stable on the LCLK leading edge when REG_WREN is active.
  - Register access valid only when USR_ACCESS = '1'.
  - Reading from a register:
  - Data must be driven through REG_DOUT and stable on the LCLK leading edge.

- Register access valid only when USR_ACCESS = '1'.